Predicting the critical temp of superconductors

Christopher Carman, Joakim Lajer, Nikolaj Andersson
NBI

All group members have contributed evenly to the project

Outline

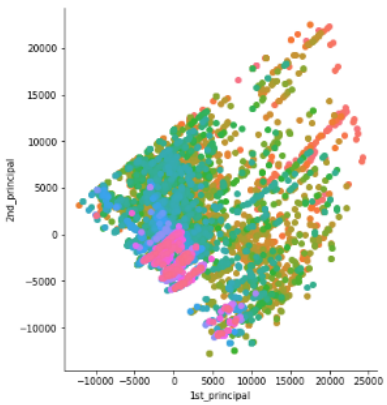
Motivation

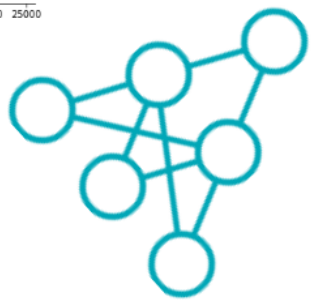
1. What is a superconductor
2. Why is it important
3. No (succesful) theory for predicting T_c

Approach

- Get data
- Kam Hamidieh - github
- Data exploration
- Feature selection
- Go crazy with ML

National Institute for Materials Science





Data

1. Data From 2018 paper
2. Based on Japanese SuperCon Database

Data Processing

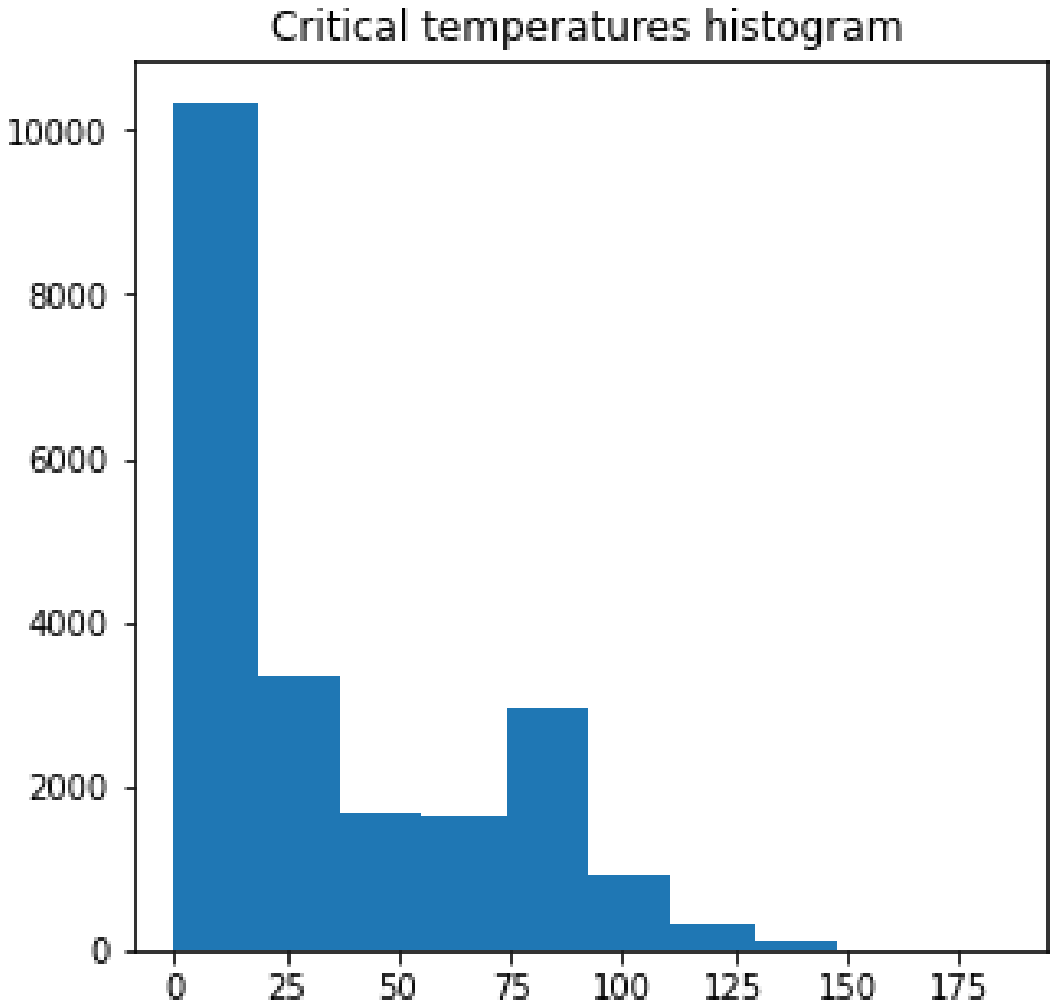
Variable	units
Atomic mass	AMU
First Ionization Energy	KJ/mol
Atomic Radius	pm
Density	kg/m^3
Electron Affinity	KJ/mol
Fusion Heat	KJ/mol
Thermal Conductivity	(W/(m x k))
Valence	NA

1. Features derived from elemental properties

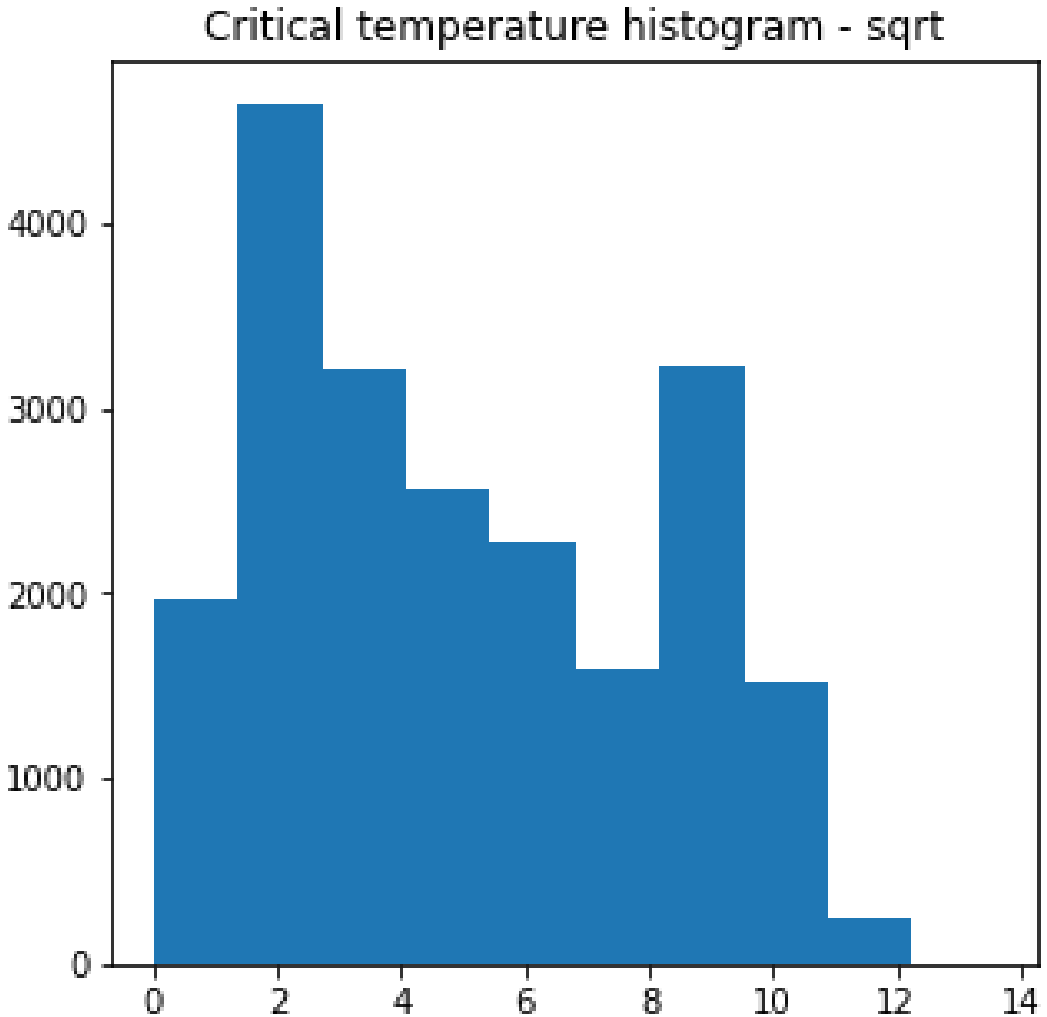
Data Features

1. Mean
2. Weighted mean
3. Geometric mean
4. Weighted geometric mean
5. Entropy
6. Weighted entropy
7. Range
8. Weighted range
9. Standard deviation
10. Weighted standard deviation

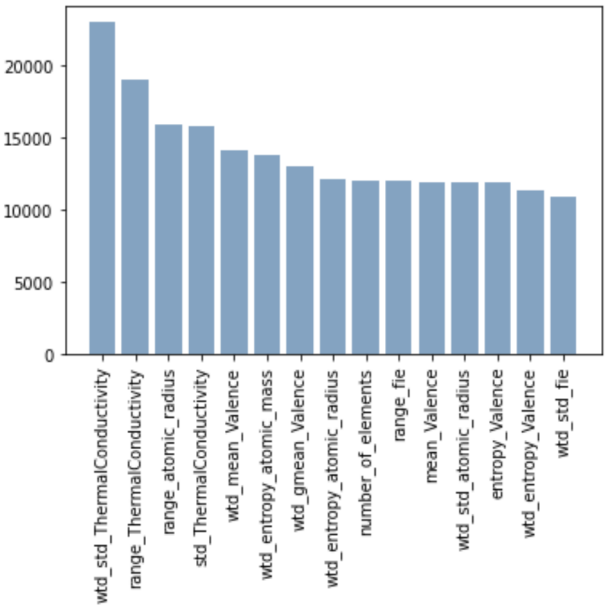
Data Exploration

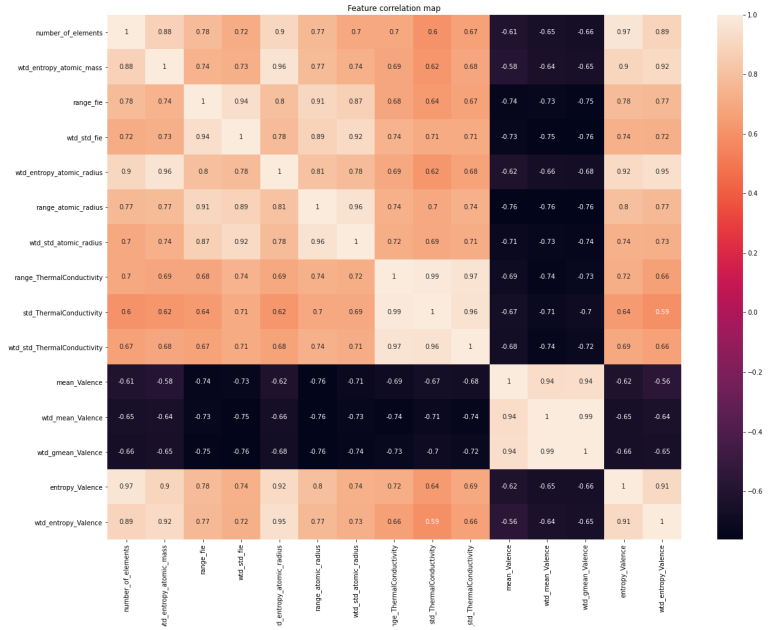


Transform


Critical temperature histogram - sqrt
4000
3000
2000
1000
0
0
2
4
6
8
10
12
14

Feature selection





Methods

1. Keras Neural Network
2. Randomized Search Tree
3. Random Forest Tree
4. Xgboost
5. K Nearest Neighbors

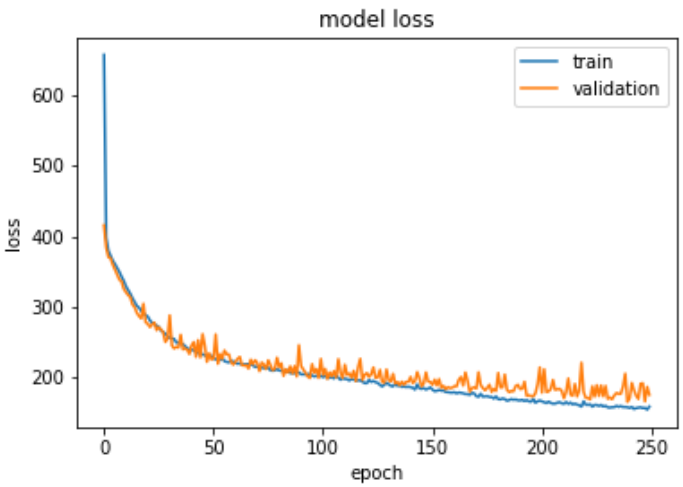
Keras Neural Network

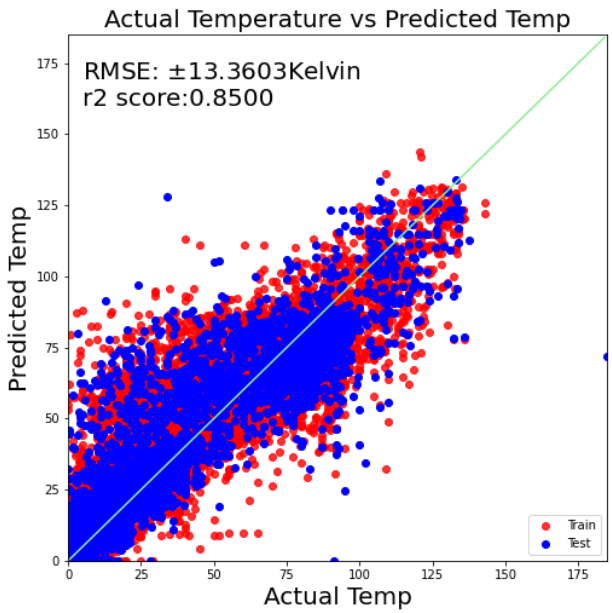
1 Model/Dimensions

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24)                384
_________________________________________________________________
dense_1 (Dense)              (None, 30)                750
_________________________________________________________________
dense_2 (Dense)              (None, 24)                744
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 25
=================================================================
Total params: 1,903
Trainable params: 1,903
Non-trainable params: 0
_________________________________________________________________
```

Keras Neural Network

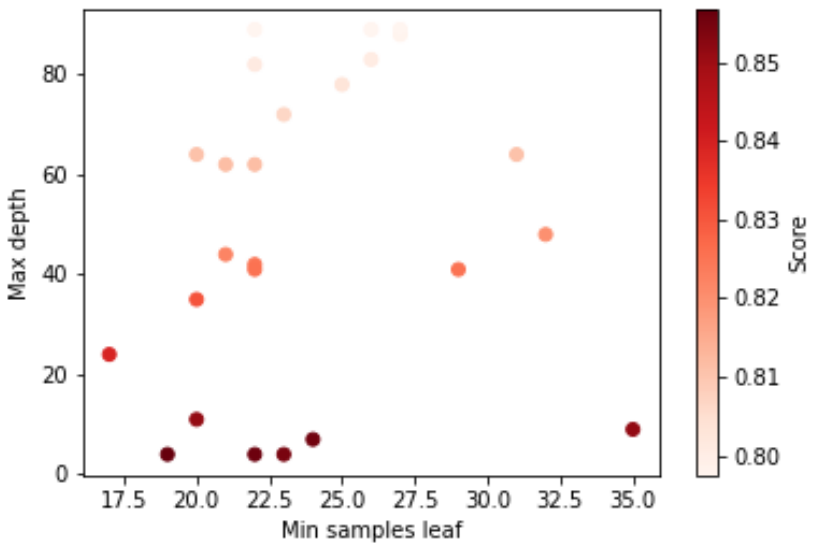
1. Efficiency/Results

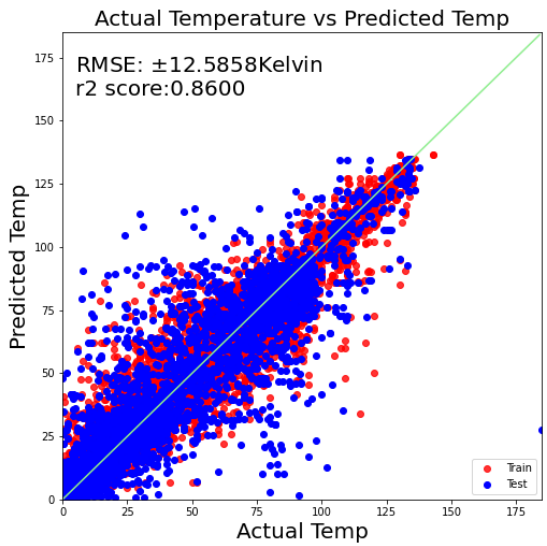




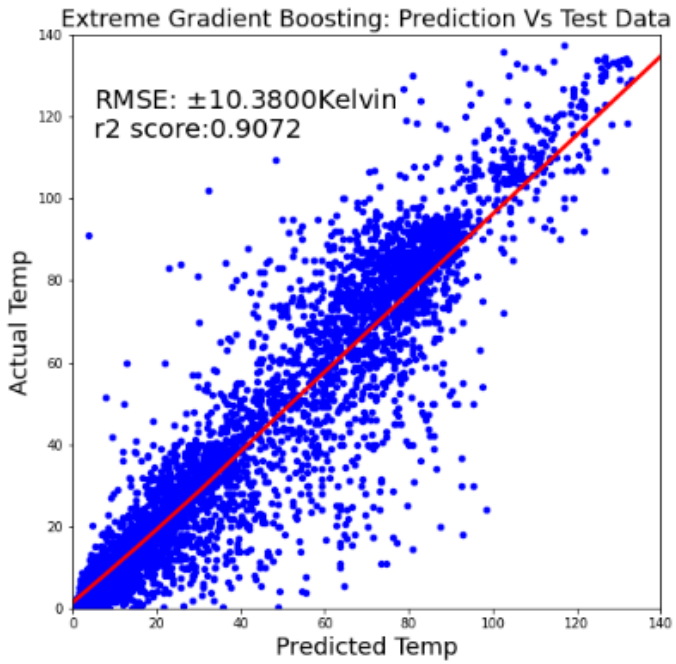
Randomized Search Tree

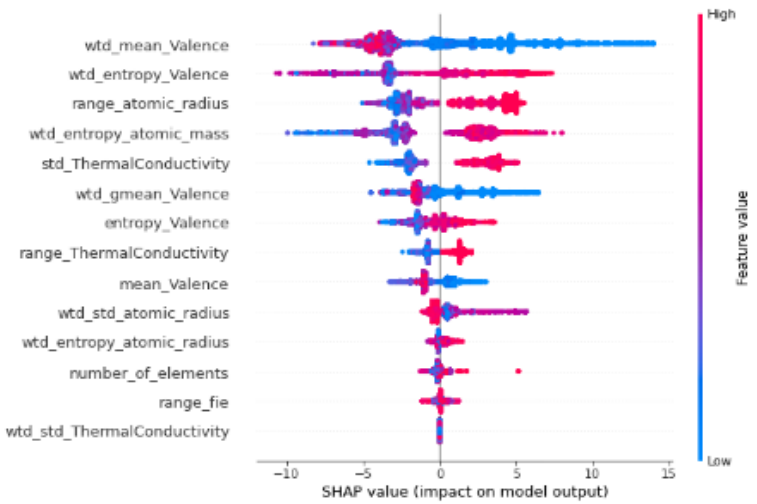
Top 5			
Max depth	Min samples leaf	Score	Rank
19	4	0.8569	1
22	4	0.8561	2
24	7	0.8556	3
23	4	0.8548	4
35	9	0.8517	5



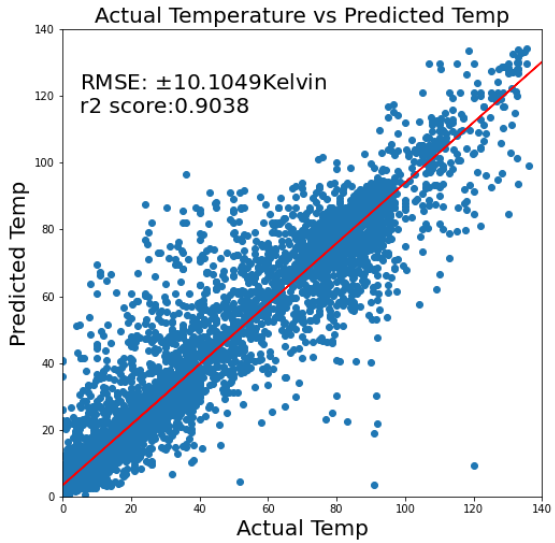


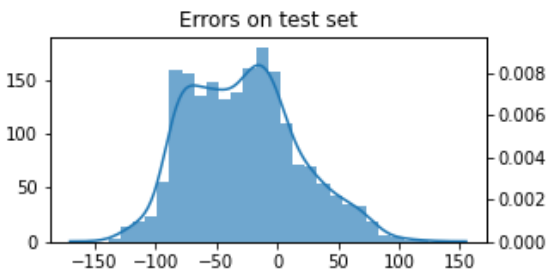
Xgboost

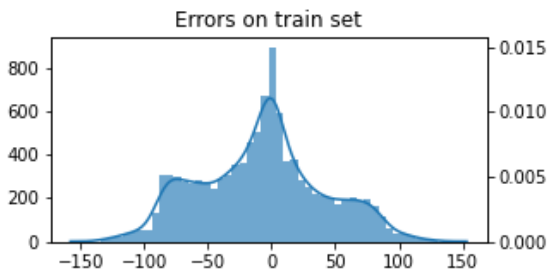

Extreme Gradient Boosting: Prediction Vs Test Data
RMSE: ±10.3800Kelvin
r2 score:0.9072
Actual Temp
Predicted Temp
0
20
40
60
80
100
120
140


wtd_mean_Valence
wtd_entropy_Valence
range_atomic_radius
wtd_entropy_atomic_mass
std_ThermalConductivity
wtd_gmean_Valence
entropy_Valence
range_ThermalConductivity
mean_Valence
wtd_std_atomic_radius
wtd_entropy_atomic_radius
number_of_elements
range_fie
wtd_std_ThermalConductivity
−10
−5
0
5
10
15
SHAP value (impact on model output)
High
Feature value
Low

Random Forest Tree

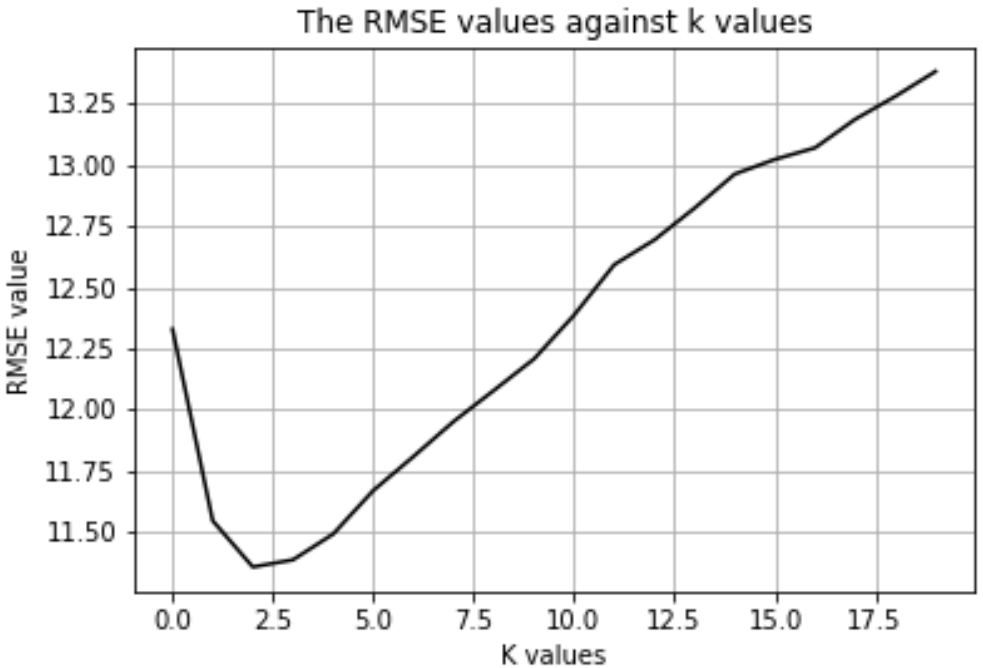




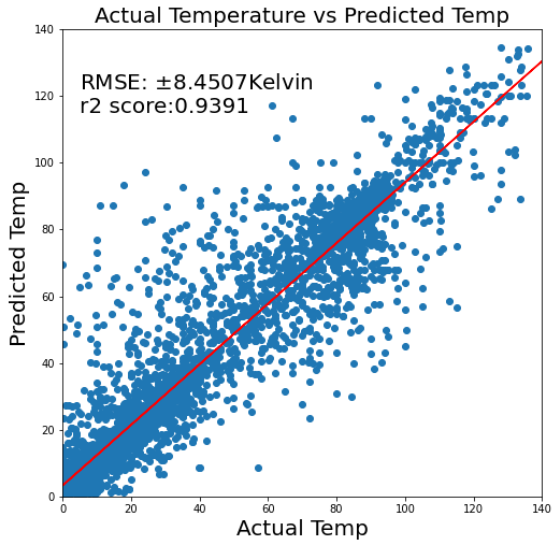


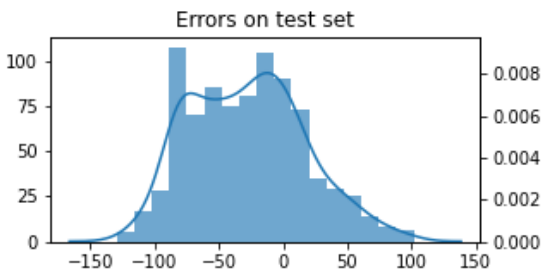
K Nearest Neighbors

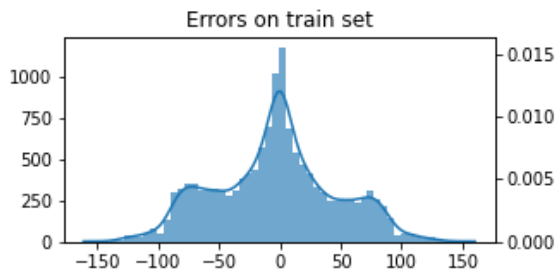
1. Finding the K value



K Nearest Neighbors

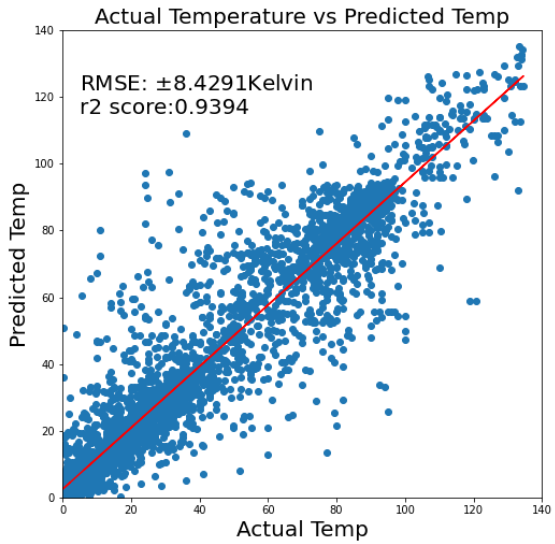


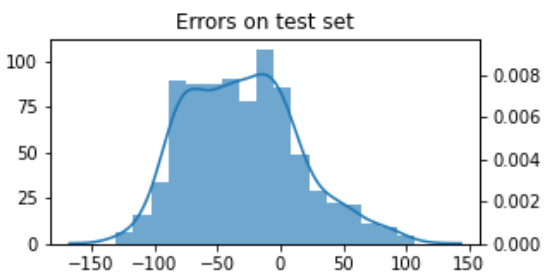


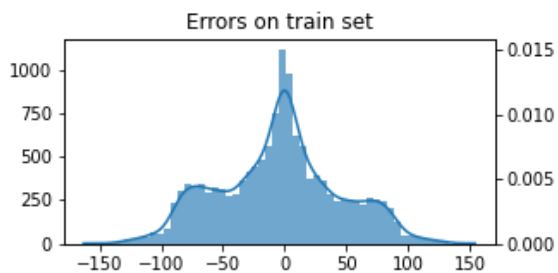


K Nearest Neighbors

1 Optimizing the algorithm







Evaluation

1. Performances of methods
2. Ranking

Method:	r2	RMSE	Speed
Keras Neural Network	0.8500	±13.3603	Medium
Randomized Search	0.8600	±12.5858	Fast
Xgboost	0.9072	±10.3800	Slow
Random Forest	0.9038	±10.1049	Fast
K Nearest Neighbors	0.9391	±8.4507	Medium
K Nearest Neighbors Optimized	0.9394	±8.4291	Semi-Slow

Thank you for your attention

References

Links to all Ipython notebooks (google colab) used in this project:

1. Data exploration
2. Feature selection
3. Keras Neural Network
4. Random Search Tree
5. XGBoost
6. Random Forrest Tree
7. kNN
8. kNN v2

Appendix - Feature selection

```
selector_f=SelectPercentile(f_regression,percentile=25)
selector_f.fit(X_var,y_var)
```

```
SelectPercentile(percentile=25,
                 score_func=<function f_regression at 0x7f6ac79c9378>)
```

```
for n,s in zip(X_var,selector_f.scores_):
    print('F-score: %3f\t %s ' % (s,n))
```

```
import numpy as np
from sklearn.feature_selection import SelectKBest

# feature extraction
test = SelectKBest(f_regression, k=15)
fit = test.fit(X_var, y_var)
```

```
from itertools import cycle, islice
# Get the indices sorted by most important to least important
indices = np.argsort(fit.scores_)[::-1]

# To get top 15 feature names
features = []
for i in range(15):
    features.append(df.columns[indices[i]])

# plot
plt.figure()
plt.xticks(rotation=90)
plt.bar(features, fit.scores_[indices[range(15)]], color=(0.2, 0.4, 0.6, 0.6), align='center')
plt.savefig('feature_selection_rankings.png', bbox_inches = "tight")
plt.show()
```

Appendix - Keras Neural Network

```
model = Sequential()
model.add(Dense(24, input_dim=X_train.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(24, activation='relu'))
model.add(Dense(1, activation='relu'))
model.summary()
```

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
```

```
r2 = round(sm.r2_score(y_test, y_pred), 2)
RMSE = sqrt(mean_squared_error(y_test,y_pred))
```

Appendix - Random Search Tree

```
clf_DecisionTree = DecisionTreeRegressor(random_state=42)
```

```
parameters_RandomSearch = {'max_depth': poisson(25),
                           'min_samples_leaf': randint(1, 100)}
```

```
n_iter_search = 25
RandomSearch = RandomizedSearchCV(clf_DecisionTree,
                                  param_distributions=parameters_RandomSearch,
                                  n_iter=n_iter_search,
                                  cv=5,
                                  iid=True,
                                  return_train_score=True,
                                  random_state=42)
```

```
clf_RandomSearch = RandomSearch.best_estimator_
```

```
y_pred = clf_RandomSearch.predict(X_test)
```

```
r2 = round(sm.r2_score(y_test, y_pred), 2)
RMSE = sqrt(mean_squared_error(y_test,y_pred))
```

Appendix - XGBoost

```
DM_train = xgb.DMatrix(data = X_train,
                       label = y_train)


DM_test =  xgb.DMatrix(data = X_test,
                       label = y_test)
```

```
gbm_param_grid = {
     'colsample_bytree': np.linspace(0.5, 0.9, 5),
     'n_estimators':[100, 200],
     'max_depth': [10, 15, 20, 25]
}
```

```
grid_mse = GridSearchCV(estimator = gbm, param_grid = gbm_param_grid,
                        scoring = 'neg_mean_squared_error', cv = 5, verbose = 1)
scores = cross_val_score(grid_mse, X, y, scoring='r2')
```

```
grid_mse.fit(X_train, y_train)

print("Best parameters found: ",grid_mse.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(grid_mse.best_score_)))
```

Appendix - Random Forrest Tree

```
scaler = MinMaxScaler()
X = df[variables]
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
y = df['critical_temp']
```

```
rf = RandomForestRegressor(n_estimators=200,max_features='log2')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)
rf.fit(X_train,y_train)

y_pred_test = pd.Series(rf.predict(X_test))
y_pred_train = pd.Series(rf.predict(X_train))

min_rmse = round(np.sqrt(mean_squared_error(y_test,y_pred_test)),4)
r2 = r2_score(y_pred_test,y_test)
```

Appendix - kNN

```
scaler = StandardScaler()
X = df[variables]
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
y = df['critical_temp']
```

```
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    knn = neighbors.KNeighborsRegressor(n_neighbors = K)

    knn.fit(X_train, y_train)  #fit the model
    pred=knn.predict(X_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ' , K , 'is:', error)
```

```
#Create KNN regressor
knn = KNeighborsRegressor(n_neighbors=3)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)
```

Appendix - KNN v2

```
scaler = MinMaxScaler()
X = df[variables]
X2 = df[variables]
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X_d = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
#print(X1.head())
y = df['critical_temp']

X = X.copy().apply(np.sqrt)
y = y.copy().apply(np.sqrt)

no_elements = df['number_of_elements']
X = pd.concat([no_elements.to_frame(), X], axis = 1)
```

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5],
          'leaf_size':np.arange(1,40)}

knn = neighbors.KNeighborsRegressor()

knn = GridSearchCV(knn, params, cv=10)
knn.fit(X_train,y_train)
knn.best_params_
```